Mercari Price Suggestion

Ashish Jain
MT2020039
International Institute of Information Technology
Bangalore
ashish.jain039@iiitb.org

Mohit Lakhotia
MT2020124
International Institute of Information Technology
Bangalore
mohit.lakhotia@iiitb.org

Ashutosh Shrivastava
MT2020139
International Institute of Information Technology
Bangalore
ashutosh.shrivastava@iiitb.org

***Abstract*— It can be hard to know how much something's really worth. Small details can mean big differences in pricing. Product pricing gets even harder at scale, considering just how many products are sold online. Clothing has strong seasonal pricing trends and is heavily influenced by brand names, while electronics have fluctuating prices based on product specs. In this application, a predictive model is built for finding the price of an item from their characteristics, especially the description. The implementation of word vectors optimized Stochastic gradient descent are used to tackle the problem.**

Index Terms—Feature Engineering, TFIDF Vectorizer, One vs Rest classifiers, Grid Search, Cross Validation, Stochastic Gradient Descent Regression.

INTRODUCTION

Setting the price for a product is getting harder at scale. This situation is getting worse in the second-hand market, that people have no idea about the value of the product they are trying to sale to achieve buyer and seller satisfaction. For example, thousands of brands, seasonal pricing trend and fashion of the clothing, the technology employed for the electronicdevice should all be considered as effective factor to set the price from human perspective.But what exact price should a seller tag on the product?

The traditional way might be searching the price of the related product, asking retailers'suggestion and make a discount from the original price. However, it will take a lot of effort,which might be more expensive compared to the product itself. In this thesis, machinelearning algorithms will be employed to tackle this problem.

The price-prediction data is obtained on Kaggle, and the provider Mercari, Japan's biggestcommunity-powered shopping app offers a price suggesting challenge to solve this problemas anyone can sell any legal product on their site1. By solving this challenge with reasonableerror, sellers will benefit a lot from saving their effort to think about the price to set and the efficiency of the market will be proved significantly.

I. RELATED WORK

It was challenging to find literature that addressed our problem. So we stitched together ideas from disparate areas including sentiment classification to develop our baseline model.

The first element that we explored was Word Embedding. We found literatures that talked about various word embeddings [1]. We decided to train using Count Vectorizer and Tf-Idf.

The second element was doing Sentiment analysis. For the same, we used VADER Algorithm [2].

Several articles and books also helped by providing a lot of useful information [3-6].

II. DATASET

The dataset used in the project is a part of the "Mercari Price Suggestion Challenge" provided by Kaggle's website Each row in the training data consists of 8 columns, which are: id, name, item condition, category name, brand name, price, shipping, item description. There are 10,37,774 datapoints in the training data and the primary test data consists of 4,44,761 datapoints for which the price is not released. Each row of the train.tsv file has the following attributes/features that lists details about a particular product.

1. $train - id$ or $test - id$ - the id of the product
2. $name$ - the name of the product
3. $item - condition - id$ - the condition of the product provided by the seller
4. $category - name$ - category of the product
5. $brand - name$: brand name of the product
6. $price$ - the price that the product was sold for. The unit is USD.
7. $shipping$ - 1 if shipping fee is paid by seller and 0 by buyer
8. $item - description$ - the full description of the item.

Input features: $train - id$, $name$, $item - condition - id$, $category - name$, $brand - name$, $shipping$, $item - description$

Target Variable: price

The most interesting part about working with this dataset is the fact that it's huge. Because of this, some sampling of the dataset is done to perform EDA a bit quicker. Python will be the language of choice.

III. OBSERVATIONS

The very first step in solving any case study in data science is to properly look and analyze the data you have. It helps to give valuable insights into the pattern and information it has to convey. Firstly, performing univariate analysis of every feature in the data set and look for hidden relationships(if any).

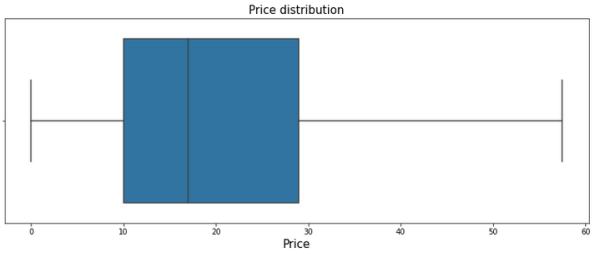
```
merdata.price.describe()
```

```
count    1.037774e+06
mean     2.673494e+01
std      3.862050e+01
min      0.000000e+00
25%      1.000000e+01
50%      1.700000e+01
75%      2.900000e+01
max      2.004000e+03
Name: price, dtype: float64
```

So, from the describe table we can conclude that:

1. 25 percent of the products are priced below 10 USD, 50 percent of products are priced below 17USD and 75USD of products are priced below 29USD.

2. Also, the maximum price that any product has is 2004USD.

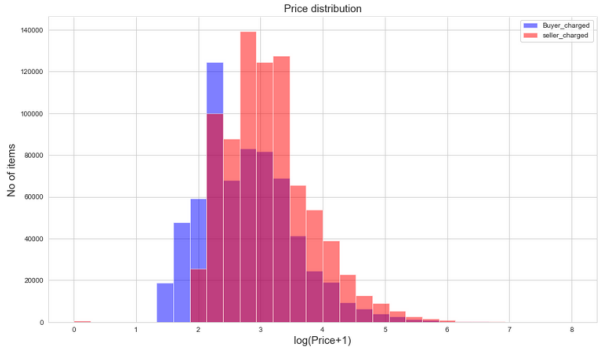


It can be seen that most of the items have the price value between 25-28.And all the items have their price listings between 10-28.

It can concluded that the distribution of the 'price' variable is heavily right-skewed. As the 'price' variable followed a skewed distribution , in order to make errors on low price product more relevant than for higher prices, the metric of evaluation in this competition is Root Mean Squared Logarithmic Error (RMSLE). Thus, log transformation is applied to the price target variable, to make this assumption available for model training. So, we have to scale down the 'price' feature to log scale.

Price VS Shipping:

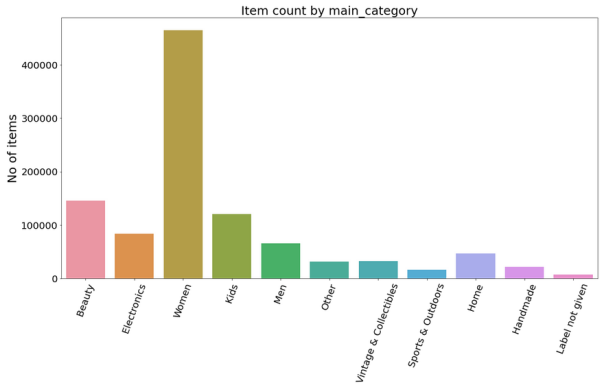
The shipping prices of products are either '1'(buyer charged) or '0'(seller charged).



Here, it can be seen that for items which have lesser price, the shipping had to be paid by the buyer for profit reasons. Also, as the price increases, we can see that the shipping charges have been paid by the seller.. And there is a lot of overlap for items where both buyer and seller have been charged. This trend is what is usually observed when we buy products online and our 'price' value is less than a certain threshold for free shipping.

Price VS Item Category: From the statistics it can be said that Women apparel has the maximum number of items followed by any other category. The category names are listed by '/' delimiter which tells about the main category, sub-category 1 and sub-category 2 of the products. Therefore, to get better idea of each product, feature engineering is done here and split the category name into 3 different columns namely, 'general cat', 'subcat1' and 'subcat2'.

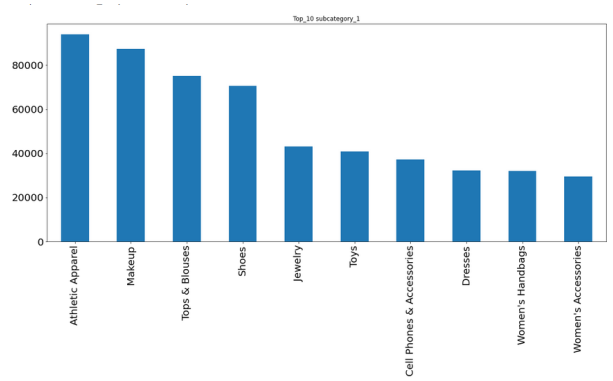
General Category:



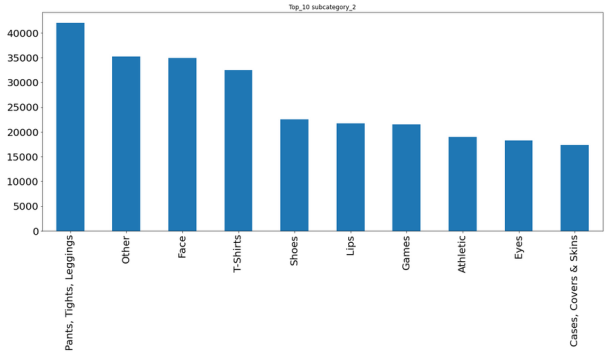
From the histogram above, it can be said that Women products occur with the maximum frequency, followed by

Beauty products. The 3rd largest general category is owned by Kids products.

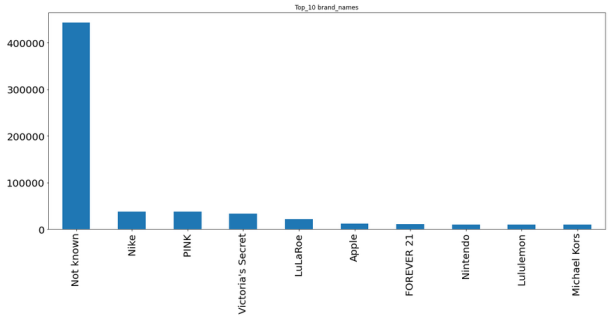
Sub-Category 1:



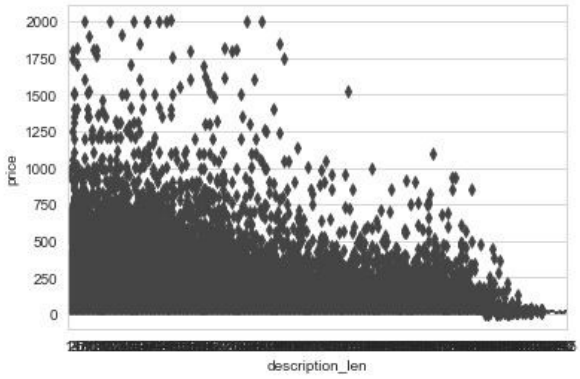
As women products occur with the maximum frequency in General category , the category with the highest frequency in Subcategory 1 is Athletic Apparel which conveys good idea about the observation in the main category. Second most popular item in the Sub category 1 is makeup which also aligns with the idea that women products occur with maximal frequency. Sub-Category 2:



It is known that Women products has the maximum frequency in General category and Athletic Apparel occurs with the highest frequency in Subcategory 1. So, this clearly aligns with the idea that 'Pants,Tights,Leggings' occur with the maximum frequency in Sub category 2. Price VS Brand name:



For the brand name column in which there was no brand information given, we have filled those cells with the value 'Not known'. And most of the items, the brand name has not been listed which can be deduced from the histogram plot. Second to it, most number of items have 'Pink' and "Nike" as brand names. Price VS Item Description Length:



Here, description length vs price has been given. From the plot, it can be said as length increases, price charged becomes lesser and lesser. Most of the items with lesser description length have more price value.

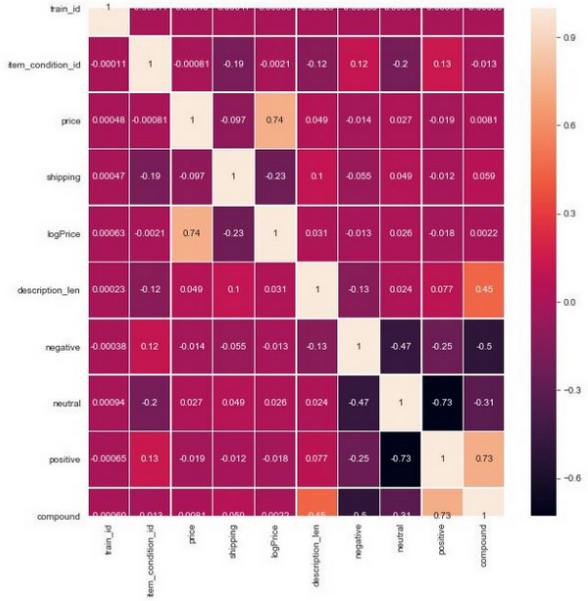
Item Category:

To understand useful patterns from the 'item-description' feature, we will visualize the data by plotting words clouds. This will help us to see the words that occur most frequently in the description column.

This word cloud plot gives a good idea about the words that appear frequently in the item description field. Above we created a word cloud with 300 most frequently occurring words. The words that occur with a larger font compared to the others have occurred more frequently than the others. Also, We can see that the columns 'category-name', 'brand-name' and 'item-description' have null values. There are different techniques of dealing with missing values like removal of the rows with values missing, remove the feature with a high percentage of missing values or fill the columns with missing values by some other values. Here, we have chosen to fill these missing values in the data by some other value.

Plotting Correlation matrix:

A correlation matrix is basically a covariance matrix that is a very good technique of multivariate exploration. There are so many features already present in our data set. In addition to that, we have created two additional features, 'item description length' and 'sentiment score of item description'. To visualize if one feature has a strong correlation with another feature, we shall analyse correlation matrix. A positive value of correlation suggests a stronger association and vice-versa.



From the above table, it can be said our newly created feature 'description-len' shares fair correlation with the target variable 'price' of an item. Also, sentiment scores shares some correlation with the 'price' variable. Hence, we will be including them as additional features in our feature list. Hence, the features that we will select for modelling are 'item-name', 'brand-name', 'shipping', 'general-category', 'subcategory-1', 'subcategory-2', 'item-description-length', 'sentiment-score-item-description' and 'item-description'.

IV. DATA PREPROCESSING AND FEATURE EXTRACTION

To make the training better we need to modify the data, which essentially means that the we need to convert the raw data we were given to more sophisticated data which would then be ready to undergo subsequent processing. Basically, we need to extract meaning out of raw data before we can use it to train our models. Models trained on meaningful data always have a upper hand to those trained on raw/less meaningful data. Hence, preprocessing the data is a very crucial step in training our model. First of all, we have used a deEmojify function to remove emoticons from item-description.

A. WORD EMBEDDINGS :

In very simplistic terms, Word Embeddings are the texts converted into numbers and there may be different numerical representations of the same text.

Why do we need Word Embeddings?

As it turns out, many Machine Learning algorithms and almost all Deep Learning Architectures are incapable of processing strings or plain text in their raw form. They require numbers as inputs to perform any sort of job, be it classification, regression etc. in broad terms. And with the huge amount of data that is present in the text format, it is imperative to extract knowledge out of it and build applications. Some real world applications of text applications are – sentiment analysis of reviews by Amazon etc., document or news classification or clustering by Google etc.

A Word Embedding format generally tries to map a word using a dictionary to a vector.

B. DIFFERENT TYPES OF WORD EMBEDDINGS :

The different types of word embeddings can be broadly classified into two categories-

1. Frequency based Embedding
2. Prediction based Embedding

Frequency based embeddings are generally three types of vectors -

1. Count Vector
2. TF-IDF Vector
3. Co-Occurrence Vector

Prediction based embeddings are generally three types of vectors -

1. CBOW
2. Skip-gram

From these, we have used Count Vectorizer and Tf-Idf Vectorizers which are discussed below.

C. COUNT VECTORIZER :

Convert a collection of text documents to a matrix of token counts.

This implementation produces a sparse representation of the counts using CSR Matrix.

If you do not provide an a-priori dictionary and you do not use an analyzer that does some kind of feature selection then the number of features will be equal to the vocabulary size found by analyzing the data.

This strategy has several advantages:

1. It is very low memory scalable to large datasets as there is no need to store a vocabulary dictionary in memory.

2. It is fast to pickle and un-pickle as it holds no state besides the constructor parameters.

3. It can be used in a streaming (partial fit) or parallel pipeline as there is no state computed during fit.

D. TF-IDF VECTORIZATION :

This is based on the frequency method but it is different to the count vectorization in the sense that it takes into account not just the occurrence of a word in a single document but in the entire corpus. So, what is the rationale behind this? Let us try to understand.

Common words like 'is', 'the', 'a' etc. tend to appear quite frequently in comparison to the words which are important to a document. For example, a document A on Lionel Messi

is going to contain more occurences of the word "Messi" in comparison to other documents. But common words like "the" etc. are also going to be present in higher frequency in almost every document.

Ideally, what we would want is to down weight the common words occurring in almost all documents and give more importance to words that appear in a subset of documents.

TF-IDF works by penalising these common words by assigning them lower weights while giving importance to words like Messi in a particular document.

So, how exactly does TF-IDF work?

Tf-Idf is a weighting scheme that assigns each term in a document a weight based on its term frequency (tf) and inverse document frequency (idf). The terms with higher weight scores are considered to be more important.

Typically, the tf-idf weight is composed by two terms-

1. Normalized Term Frequency (tf)

2. Inverse Document Frequency (idf)

Step 1: Computing the Term Frequency(tf) Frequency indicates the number of occurences of a particular term t in document d. Therefore,

tf(t, d) = N(t, d), wherein tf(t, d) = term frequency for a term t in document d.

N(t, d) = number of times a term t occurs in document d

Step 2: Compute the Inverse Document Frequency – idf

It typically measures how important a term is. The main purpose of doing a search is to find out relevant documents matching the query. Since tf considers all terms equally important, thus, we can't only use term frequencies to calculate the weight of a term in the document. However, it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scaling up the rare ones. Logarithms helps us to solve this problem.

First of all, find the document frequency of a term t by counting the number of documents containing the term:

df(t) = N(t)

where- df(t) = Document frequency of a term t N(t) = Number of documents containing the term t

It's expected that the more frequent term to be considered less important, but the factor (most probably integers) seems too harsh. Therefore, we take the logarithm (with base 2) of the inverse document frequencies. So, the idf of a term t becomes :

idf(t) = log(N/ df(t))

Tf-Idf also has several advantages-

1. Easy to compute

2. You have some basic metric to extract the most descriptive terms in a document

3. You can easily compute the similarity between 2 documents using it

E. SENTIMENT ANALYSIS USING VADER :

Sentiment analysis is a text analysis method that detects polarity (e.g. a positive or negative opinion) within the text, whether a whole document, paragraph, sentence, or clause.

Sentiment analysis aims to measure the attitude, sentiments, evaluations, attitudes, and emotions of a speaker/writer based on the computational treatment of subjectivity in a text.

VADER

VADER (Valence Aware Dictionary for Sentiment Reasoning) is a model used for text sentiment analysis that is sensitive to both polarity (positive/negative) and intensity (strength) of emotion. It is available in the NLTK package and can be applied directly to unlabeled text data.

VADER sentimental analysis relies on a dictionary that maps lexical features to emotion intensities known as sentiment scores. The sentiment score of a text can be obtained by summing up the intensity of each word in the text.

For example- Words like 'love', 'enjoy', 'happy', 'like' all convey a positive sentiment. Also VADER is intelligent enough to understand the basic context of these words, such as "did not love" as a negative statement. It also understands the emphasis of capitalization and punctuation, such as "ENJOY".

VADER's SentimentIntensityAnalyzer() takes in a string and returns a dictionary of scores in each of four categories:

negative neutral positive compound (computed by normalizing the scores above)

The results of VADER analysis don't seem to be only remarkable but also very encouraging. The results show the advantages which will be attained by the utilization of VADER in cases of web sites wherein the text data could be a complex mixture of a range of text.

V. MODEL SELECTION

After having done with the statistical analysis and cleaning of data, we also did some feature engineering and added two new features- 'length of item description' and 'sentiment score of an item' from the already existing features. We also handled categorical, text and numerical features and merged into train and test data matrix. We are now ready to apply machine learning algorithms on our prepared data matrices.

There are many machine learning algorithms to try when you are working on a specific problem. You never know which ML model would work the best for you before hand. However, domain knowledge and expertise with practice do play a role sometimes. But the best choice for anybody who wants to master his skills is by experimenting. So, we experimented with a couple of regression models and checked which performs the best.

Initially, we tried to use Linear Regression to predict price values. Then we experimented with three machine learning regression models:

1. Ridge Regression

2. Stochastic Gradient Descent Regression

3. Light Gradient Boosting Regression

A. LINEAR REGRESSION

Linear regression attempts to model the relationship between two variables by fitting a linear equation to observed data. One variable is considered to be an explanatory variable, and the other is considered to be a dependent variable.

For example, a modeler might want to relate the weights of individuals to their heights using a linear regression model.

But due to accuracy issues(0.78 score), we had to switch to other models.

B. STOCHASTIC GRADIENT DESCENT

Stochastic gradient descent (often abbreviated SGD) is an iterative method for optimizing an objective function with suitable smoothness properties (e.g. differentiable or subdifferentiable).

It can be regarded as a stochastic approximation of gradient descent optimization, since it replaces the actual gradient (calculated from the entire data set) by an estimate thereof (calculated from a randomly selected subset of the data). Especially in high-dimensional optimization problems this reduces the computational burden, achieving faster iterations in trade for a lower convergence rate.

We used this model because of following reasons -

1. On massive datasets, stochastic gradient descent can converges faster because it performs updates more frequently. Also, the stochastic nature of online/minibatch training takes advantage of vectorised operations and processes the mini-batch all at once instead of training on single data points.

2. Performing one pass of SGD on a particular dataset is statistically (minimax) optimal. In other words, no other algorithm can get one better results on the expected loss.

3. Stochastic gradient descent delivers similar guarantees to empirical risk minimisation, which exactly minimises an empirical average of the loss on training data.

C. RIDGE REGRESSION

Ridge Regression is a technique for analyzing multiple regression data that suffer from multicollinearity. When multicollinearity occurs, least squares estimates are unbiased, but their variances are large so they may be far from the true value. By adding a degree of bias to the regression estimates, ridge regression reduces the standard errors.

It is hoped that the net effect will be to give estimates that are more reliable. Another biased regression technique, principal components regression, is also available in NCSS.

Ridge regression is the more popular of the two methods.

The ridge regression has two important advantages over the linear regression.

The most important one is that it penalizes the estimates. It doesn't penalize all the feature's estimate arbitrarily. If estimates (β) value are very large, then the SSE term in the above equation will minimize, but the penalty term will increases. If estimates (β) values are small, then the penalty term in the above equation will minimize, but, the SSE term will increase due to poor generalization. So, it chooses the feature's estimates (β) to penalize in such a way that less influential features (Some features cause very small influence on dependent variable) undergo more penalization. In some domains, the number of independent variables is many, as well as we are not sure which of the independent variables influences dependent variable. In this kind of scenario, ridge regression plays a better role than linear regression.

Another advantage of ridge regression over OLS is when the features are highly correlated with each other, then the rank of matrix X will be less than P+1 (where P is number of regressors). So, the inverse of $X^T.X$ doesn't exist, which results to OLS estimate may not be unique.

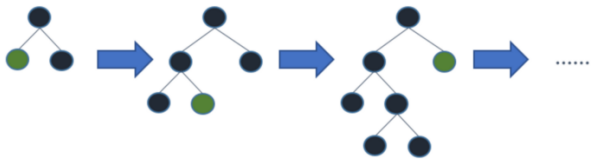
For ridge regression, we are adding a small term k along the diagonals of $X^T.X$. It makes the $X^T.X + k.I$ matrix to be invertible (All the columns are linearly independent).

D. LGBM

Light GBM is a gradient boosting framework that uses tree based learning algorithm. How it differs from other tree based algorithm?

Light GBM grows tree vertically while other algorithm grows trees horizontally meaning that Light GBM grows tree leaf-wise while other algorithm grows level-wise. It will choose the leaf with max delta loss to grow. When growing the same leaf, Leaf-wise algorithm can reduce more loss than a level-wise algorithm.

Below diagrams explain the implementation of LightGBM and other boosting algorithms.



Leaf-wise tree growth

Level-wise tree growth

The size of data is increasing day by day and it is becoming difficult for traditional data science algorithms to give faster results. Light GBM is prefixed as 'Light' because of its high speed. Light GBM can handle the large size of data and takes lower memory to run. Another reason of why Light GBM is popular is because it focuses on accuracy of results. LGBM also supports GPU learning and thus data scientists are widely using LGBM for data science application development.

Implementation of Light GBM is easy, the only complicated thing is parameter tuning.

Parameters

Control Parameters

max-depth: It describes the maximum depth of tree. This parameter is used to handle model overfitting. Any time you feel that your model is overfitted, my first advice will be to lower max-depth.

min-data-in-leaf: It is the minimum number of the records a leaf may have. The default value is 20, optimum value. It is also used to deal over fitting

feature-fraction: Used when your boosting(discussed later) is random forest. 0.8 feature fraction means LightGBM will select 80 percent of parameters randomly in each iteration for building trees.

bagging-fraction: specifies the fraction of data to be used for each iteration and is generally used to speed up the training and avoid overfitting.

early-stopping-round: This parameter can help you speed up your analysis. Model will stop training if one metric of one validation data doesn't improve in last early-stopping-round rounds. This will reduce excessive iterations.

lambda: lambda specifies regularization. Typical value ranges from 0 to 1.

min-gain-to-split: This parameter will describe the minimum gain to make a split. It can used to control number of useful splits in tree.

max-cat-group: When the number of category is large, finding the split point on it is easily over-fitting. So LightGBM merges them into 'max-cat-group' groups, and finds the split points on the group boundaries, default:64

Core Parameters

Task: It specifies the task you want to perform on data. It may be either train or predict.

application: This is the most important parameter and specifies the application of your model, whether it is a regression problem or classification problem. LightGBM will by default consider model as a regression model.
regression: for regression
binary : for binary classification
multiclass: for multiclass classification problem

boosting: defines the type of algorithm you want to run, default=gdbt
gbdt: traditional Gradient Boosting Decision Tree
rf: random forest
dart: Dropouts meet Multiple Additive Regression Trees
goss: Gradient-based One-Side Sampling

num-boost-round: Number of boosting iterations, typically 100+

learning-rate: This determines the impact of each tree on the final outcome. GBM works by starting with an initial estimate which is updated using the output of each tree. The learning parameter controls the magnitude of this change in the estimates. Typical values: 0.1, 0.001, 0.003...

num-leaves: number of leaves in full tree, default: 31

device: default: cpu, can also pass gpu

Metric parameter

metric: again one of the important parameter as it specifies loss for model building. Below are few general losses for regression and classification.
mae: mean absolute error
mse: mean squared error
binary-logloss: loss for binary classification
multi-logloss: loss for multi classification

IO parameter

max-bin: it denotes the maximum number of bin that feature value will bucket in.

categorical-feature: It denotes the index of categorical features. If categorical-features=0,1,2 then column 0, column 1 and column 2 are categorical variables.

ignore-column: same as categorical-features just instead of considering specific columns as categorical, it will completely ignore them.

save-binary: If you are really dealing with the memory size of your data file then specify this parameter as 'True'. Specifying parameter true will save the dataset to binary file, this binary file will speed your data reading time for the next time.

Ensemble:
The final step was to figure out which model gave the best results. We are using Ensemble modelling on all the 4 regression based algorithms that we have used for prediction to be used in our final submission file. Since RMSLE is the indicator metric for our problem, we will be giving weights to each of the algorithms based on their RMSLE values. Since, LightGBM gives the best results and hence we gave maximum weight-age to it while calculating the final predicted price of an item in ensemble modelling.

VI. CONCLUSION

This was our first self-case study on Machine Learning and also my first Kaggle competition submission. We got to learn tonnes of techniques while working to improve accuracy on the machine learning modelling. It always feels great to read from books and blogs about the methodologies, but unless you do it on your own and learn things from scratch, you won't get a good idea about how you can solve this in practical.

This concludes our work.

CHALLENGES AND FUTURE SCOPE

The above table is the summary on the train.csv data set. But the evaluation in the competition would be on their submission data set. So, we tried submitting using the best hyper parameters from each of these models. But the key point to note that almost all the individual ML models had comparatively higher values of RMSLE, whereas model generated by using Ensemble modelling gave the lowest RMSLE values and hence is my final model for this case study.

Improvements to Existing Approaches:

We have tried feature engineering and included two new features : 'length of item description' and 'sentiment score of an item' in our modelling. Of all the online solutions that we have went through, most of them actually did not add 'sentiment score of an item' as a feature engineering technique which we think is an improvement on our way of solving the problem. We have also extensively hyperparameter tuned all my models with a huge set of feature combinations which is definitely an improvement over online solutions.

Future Work:

As a scope of future work, we wish to engineer more features and wish to improve the accuracy of my models.

We also wish to experiment with more hyperparameters and try different machine learning algorithms to further improve our solutions.

ACKNOWLEDGEMENT

We would like to thank Professor G. Srinivas Raghavan and our Machine Learning Teaching Assistants - Amitesh Anand, Tejas Kotha and Tanmay Jain for giving us the opportunity to work on the project and help us whenever we were struck by giving us ideas and resources to learn from. A special thanks to Team Crash Test Dummies(Gourav Sachdev, Yashasvi Khandelwal and Anshul Garg) for having a healthy discussions regarding various approaches in pre-processing, feature selection, algorithm optimizations etc.

REFERENCES

[1] https://www.analyticsvidhya.com/blog/2017/06/word-embeddings-count-word2veec/
[2] https://towardsdatascience.com/sentimental-analysis-using-vader-a3415fef7664
[3] https://scikit-learn.org/stable/
[4] Joulin, A.,grave, E., Bojanowski, P. Mikolov, T.(2016). Bag of tricks for efficient text classifications arXiv preprint arXiv:1607.01759
[5] Levy, O., Goldberg, Y., and Dagn, I.(2015). Improving distributional similarity with lessons lerned from word embeddings. TACL
[6] Pryzant, R., Chung, Y.J., Jurafsky, D.(2017). Predicting sales from the Language of Products Descriptions
[7] https://scikit-learn.org/stable/modules/generated/sklearn.feature-extraction.text.TfidfVectorizer.html